

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

11.

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平8-185340

(43) 公開日 平成8年(1996)7月16日

(51) Int.Cl.⁶

G 0 6 F 11/28
9/38

識別記号

3 1 5 A 7313-5B
3 8 0 C

庁内整理番号

F I

技術表示箇所

審査請求 未請求 請求項の数 4 O L (全 21 頁)

(21) 出願番号 特願平6-326400

(22) 出願日 平成6年(1994)12月27日

(71) 出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72) 発明者 久保田 和人

神奈川県川崎市幸区小向東芝町1番地 株
式会社東芝研究開発センター内

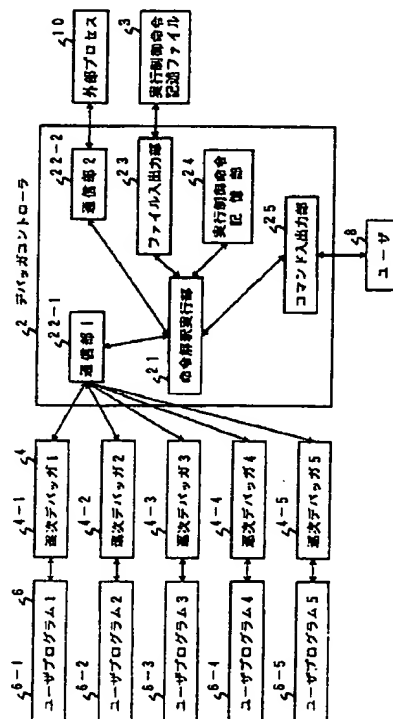
(74) 代理人 弁理士 鈴江 武彦

(54) 【発明の名称】 並列プログラムデバッグ方法および並列プログラム可視化装置

(57) 【要約】

【目的】 複数のプログラムの動作状態を関連付けたデバッグを行うことのできる並列プログラムデバッグ方法を提供することを目的とする。

【構成】 並列プログラムを構成する複数の逐次プログラムに一对一に対応して設けられた複数の逐次デバッグ夫々をデバッグコントローラが制御して並列プログラムのデバッグを行う並列プログラムデバッグ方法において、デバッグコントローラは、外部から与えられた並列プログラムに対するデバッグ内容の記述を解釈し、この解釈結果に従いデバッグ夫々に命令を与えて並列プログラムの実行を制御させデバッグ夫々から実行制御結果を取得し、これら実行制御結果に基づいてデバッグ夫々にさらに命令を与えて並列プログラムの実行を制御させデバッグ夫々から実行制御結果を取得する手続きを少なくとも1回行い、実行制御結果に従って生成したデバッグ結果を外部に出力することを特徴とする。



【特許請求の範囲】

【請求項1】 並列プログラムを構成する複数の逐次プログラムに一つ一つに対応して設けられた複数の逐次デバッグ夫々をデバッグコントローラが制御して該並列プログラムのデバッグを行う並列プログラムデバッグ方法において、

前記デバッグコントローラは、
外部から与えられた並列プログラムに対するデバッグ内容の記述を解釈し、

この解釈結果に従い、前記デバッグ夫々に命令を与えて、前記並列プログラムの実行を制御させ、該デバッグ夫々から実行制御結果を取得し、

これら実行制御結果に基づいて、前記デバッグ夫々にさらに命令を与えて、前記並列プログラムの実行を制御させ、該デバッグ夫々から実行制御結果を取得する手続きを少なくとも1回行い、

前記実行制御結果に従って生成したデバッグ結果を外部に出力することを特徴とする並列プログラムデバッグ方法。

【請求項2】 並列プログラムを構成する複数の逐次プログラムに一つ一つに対応して設けられた複数の逐次デバッグと、これら複数の逐次デバッグ夫々を制御して該並列プログラムのデバッグを行うデバッグコントローラと、該デバッグコントローラからの命令によってデバッグ結果の表示を行う少なくとも1つの表示手段とを具備してなる並列プログラム可視化装置において、

前記デバッグコントローラは、
取得すべき少なくとも1つの変数および該変数を取得する前記逐次プログラム中の位置を含む変数獲得情報を記憶する手段と、

前記逐次デバッグ夫々に命令を与えて、前記並列プログラムの実行を制御させ、前記変数獲得情報に基づき、該逐次デバッグ夫々から前記変数の値を取得する手段と、
所定のタイミングで、前記変数の値を指定された前記表示手段に与える手段とを有し、

前記表示手段は、
デバッグ結果を表示するために前記変数のうち少なくとも1つを用いて作成された表示式を記憶する手段と、
前記デバッグコントローラから与えられた前記変数の値を前記表示式に基づいて表示する手段とを有することを特徴とする並列プログラム可視化装置。

【請求項3】 前記デバッグコントローラは、
前記変数の値のうち少なくとも1つを加工するための簡易言語を記憶する手段と、

所定のタイミングで、前記簡易言語に従って、前記逐次デバッグ夫々から取得した前記変数の値を加工する手段とをさらに具備したことを特徴とする請求項2に記載の並列プログラム可視化装置。

【請求項4】 前記表示手段は、
表示画面上に表示した図形の座標と該図形を表示するた

めの前記変数を取得した前記逐次プログラムの番号を対応付けて記憶する手段と、

前記逐次プログラムの番号を並べたプロセッサマップを表示する手段と、

所定のポインティングデバイスにより前記画面上の特定の図形が指示された場合、該指示された図形の座標と前記記憶する手段に記憶された内容から、該指示された図形を表示するための前記変数を取得した前記逐次プログラムの番号を求め、前記プロセッサマップ上の番号のうち、求められた番号の表示を変化させる手段とをさらに具備したことを特徴とする請求項2に記載の並列プログラム可視化装置。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明は、並列プログラムをデバッグするための並列プログラムデバッグ方法、および並列プログラムを可視化するためのツールである並列プログラム可視化装置に関する。

【0002】

【従来の技術】

(1) 逐次プログラムのデバッグ時には、逐次プログラム用デバッグを用いて、ブレークポイントを設定し、逐次プログラムを実行させ、プログラムが停止した時点での変数の値を読み出すという方法が用いられることが多い。

【0003】 同一のプログラムが互いに通信を行いながら複数同時に実行されるデータパラレルプログラムのデバッグの際にも、上記の逐次プログラムのデバッグの方法と同様な方法を用いる試みがなされている。これは、データパラレルプログラムを構成する個々の部分プログラム毎にデバッグを適用してブレークポイントを設定し、それぞれの部分プログラムを実行させ、個々のデバッグが停止した時点での変数の値を読み出すという方法である。しかしながら、この方法の場合、ユーザは個々のデバッグに対してデバッグコマンドを送り、結果を受け取るという作業を強いられる。もちろん、複数のプログラムが、互いに異なるものである場合にも同様である。

【0004】 この煩雑な作業の軽減を目的として、個々のデバッグを統合するコントロールプロセスを置き、ユーザはコントロールプロセスを通じて個々のデバッグにデバッグコマンドを送り、結果を受け取るという方法が、Expressoのndbという並列プログラムデバッグでは用いられている。

【0005】 ところが、ユーザがデバッグ時に並列プログラムを停止させる場合、個々の部分プログラム間で、ある一定の条件が満たされた場合に、はじめて停止させたいという場合がある。この点について、図3のプログラムを例に説明する。ここでは、図3のプログラムを5つ並列に実行させる場合について考える。いま、ユーザ

はそれぞれのプログラムを逐次型のデバッガを用いて起動し、02行目と04行目にブレークポイントを設定したものとす。それぞれのブレークポイント番号を1, 2とする。図3の02行目と04行目につけられた1, 2は、それぞれ02行目と04行目にブレークポイント番号1と2のブレークポイントが付けられたことを表す。続いてユーザは、5つの逐次デバッガに個々のプログラムの実行命令を出す。これにより、個々のプログラムは、変数yの値に応じて02行目の文か04行目の文のいずれかの文で停止する。ここで、ユーザが5つの逐次デバッガに個々のプログラムに対する継続命令を出すと、個々のプログラムはループを繰り返し実行し、再度02行目の文か04行目の文のいずれかの文で停止する。再び、ユーザが5つの逐次デバッガに個々のプログラムに対する継続命令を出すと、個々のプログラムはループを繰り返し実行し、再度02行目の文か04行目の文のいずれかの文で停止する。

【0006】さて、この一連の処理において、ユーザ側は5つのプログラムのうち3つのプログラムが02行目で停止し、2つのプログラムが04行目で停止している状態のとき、はじめて5つのプログラムを停止させたいという要求があったとする。しかし、従来の並列プログラムデバッガでは、各デバッガが、対応しているプログラムを夫々制御するだけであり、複数のプログラム間である条件が成立した場合に、例えば上記のような状態で、自動的にプログラムを停止させることはできなかった。

【0007】(2) 一方、並列プログラムでは、並列プログラムを構成する個々の部分プログラムが相互に通信を行いながら処理が進められて行くため、挙動が複雑であり、デバッグが困難である。このため、プログラムの挙動を可視化してユーザに示す試みがなされている。通常は、プログラム中に可視化のための関数を埋め込んで表示を行わせる方法がとられる。

【0008】しかしながら、可視化のための関数をプログラムに埋め込む方法では、以下のような不具合があった。まず、可視化のためのコードをユーザが記述する必要があるため処理が煩雑であった。また、ある観点に基づく可視化のためのコードを記述しても、別の観点から可視化を行いたい場合には、可視化部分を書き直す必要があった。さらに、手間をかけて記述した可視化ルーチンもデバッグ終了後には不必要なものとなる。したがって、一般ユーザにとって、簡易に可視化を行えるツールが待望されている。

【0009】

【発明が解決しようとする課題】

(1) 以上説明したように、従来は、並列プログラムをデバッグする際、例えば複数のプログラム間である条件が成立した場合に、自動的にプログラムを停止させるなど、複数のプログラムの動作状態を関連付けたデバッグ

を行うことはできなかった。

【0010】本発明は、上記事情を考慮してなされたものであり、複数のプログラムの動作状態を関連付けたデバッグを行うことのできる並列プログラムデバッグ方法を提供することを目的とする。

【0011】(2) 以上説明したように、従来は、並列プログラムを可視化する場合、可視化のための関数などをプログラムに埋め込む作業が煩雑である上に、汎用性がなく、また埋め込んだ可視化ルーチンはデバッグ終了後には不必要なものとなる不具合があった。

【0012】本発明は、上記事情を考慮してなされたものであり、並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる並列プログラム可視化装置を提供することを目的とする。

【0013】

【課題を解決するための手段】

(1) 本発明は、並列プログラムを構成する複数の逐次プログラムに一对一に対応して設けられた複数の逐次デバッガ夫々をデバッガコントローラが制御して該並列プログラムのデバッグを行う並列プログラムデバッグ方法において、前記デバッガコントローラは、外部から与えられた並列プログラムに対するデバッグ内容の記述を解釈し、この解釈結果に従い、前記デバッガ夫々に命令を与えて、前記並列プログラムの実行を制御させ、該デバッガ夫々から実行制御結果を取得し、これら実行制御結果に基づいて、前記デバッガ夫々にさらに命令を与えて、前記並列プログラムの実行を制御させ、該デバッガ夫々から実行制御結果を取得する手続きを少なくとも1回行い、前記実行制御結果に従って生成したデバッグ結果を外部に出力することを特徴とする。

【0014】好ましくは、前記デバッグ内容は、停止したプログラムの番号、各プログラムのブレークポイント、ブレークポイントでの停止時のプログラム内で使われている変数の値、当該デバッガコントローラ内で保持されている内部変数の値のうちの少なくとも1つを用いて記述しても良い。

【0015】また、好ましくは、逐次プログラムのある時点での変数の値をデバッガコントローラ内部で保持し、その逐次プログラムの変数の値がプログラムの継続実行によって破壊された後でも、デバッガコントローラのコマンドから利用することができるようにしても良い。

【0016】また、好ましくは、ブレークポイント間に階層関係を定義することで、複数のブレークポイントに関わる停止条件などの実行制御条件を簡便に設定することができるようにしても良い。

【0017】また、好ましくは、ユーザの誤ったブレークポイントの設定などによって、上記複数の逐次プログラムの全てがブレークポイントに到達することなしに停止した場合、あるいはプログラムの制御がデッドロック

を起こした場合などに、デバッガコントローラに対してユーザから割込みをかけ、制御をユーザ側に戻すことができるようにしても良い。その際、例えばブレークポイントで停止した以外の上記複数の逐次プログラムの停止位置などの情報をユーザに対して示すようにして、デッドロックが起こった原因あるいは状況を把握できるようにしても良い。

【0018】(2) 本発明は、並列プログラムを構成する複数の逐次プログラムに一つ一つに対応して設けられた複数の逐次デバッガと、これら複数の逐次デバッガ夫々を制御して該並列プログラムのデバッグを行うデバッガコントローラと、該デバッガコントローラからの命令によってデバッグ結果の表示を行う少なくとも1つの表示手段とを具備してなる並列プログラム可視化装置において、前記デバッガコントローラは、取得すべき少なくとも1つの変数および該変数を取得する前記逐次プログラム中の位置を含む変数獲得情報を記憶する手段と、前記逐次デバッガ夫々に命令を与えて、前記並列プログラムの実行を制御させ、前記変数獲得情報に基づき、該逐次デバッガ夫々から前記変数の値を取得する手段と、所定のタイミングで、前記変数の値を指定された前記表示手段に与える手段とを有し、前記表示手段は、デバッグ結果を表示するために前記変数のうち少なくとも1つを用いて作成された表示式を記憶する手段と、前記デバッガコントローラから与えられた前記変数の値を前記表示式に基づいて表示する手段とを有することを特徴とする。

【0019】好ましくは、前記デバッガコントローラは、前記変数の値のうち少なくとも1つを加工するための簡易言語を記憶する手段と、所定のタイミングで、前記簡易言語に従って、前記逐次デバッガ夫々から取得した前記変数の値を加工する手段とをさらに具備したことを特徴とする。

【0020】また、好ましくは、前記表示手段は、表示画面上に表示した図形の座標と該図形を表示するための前記変数を取得した前記逐次プログラムの番号を対応付けて記憶する手段と、前記逐次プログラムの番号を並べたプロセッサマップを表示する手段と、所定のタイミングデバイスにより前記画面上の特定の図形が指示された場合、該指示された図形の座標と前記記憶する手段に記憶された内容から、該指示された図形を表示するための前記変数を取得した前記逐次プログラムの番号を求め、前記プロセッサマップ上の番号のうち、求められた番号の表示を変化させる手段とをさらに具備したことを特徴とする。

【0021】

【作用】

(1) 本発明では、並列プログラムに対する所望のデバッグ内容をファイルなどに記述して、外部からデバッガコントローラに与えるようにしている。デバッガコントローラでは、この記述を解釈しその結果に従い、各デバ

ッガに命令を与えて並列プログラムを制御させ、実行制御結果を取得する。

【0022】そして、取得した各逐次プログラムの実行制御結果に基づいて、さらに必要な命令を与えて再び並列プログラムを制御させ、各デバッガから実行制御結果を取得する。この手続きは、必要に応じて繰り返される。

【0023】デバッガコントローラは、実行制御結果に応じて、外部に与えるデバッグ結果を作成し出力する。したがって本発明によれば、複数の逐次プログラム相互間の動作状態を関連付けて、逐次プログラムの実行制御および変数値の取得など所望のデバッグを行うことができる。

【0024】例えば、複数の逐次プログラム間である条件が成立した場合に、自動的にプログラムを停止させるような処理などが可能となる。

(2) 本発明では、デバッガコントローラ内に、取得すべき少なくとも1つの変数および該変数を取得する前記逐次プログラム中の位置を含む変数獲得情報を予め設定できるようにしている。デバッガコントローラは、各逐次デバッガに命令を与えて並列プログラムを制御させ、変数獲得情報に基づき逐次デバッガ夫々から変数の値を取得していく。そして、所定のタイミングで、取得した変数の値を指定された表示手段に与える。

【0025】表示手段内には、デバッグ結果を表示するために、表示すべき変数や該変数を用いた関数を定義した表示式を予め設定しておく。表示手段は、デバッガコントローラから与えられた変数の値をこの表示式に従って表示する。

【0026】したがって本発明によれば、各逐次デバッガを通じて取得する変数に関する情報や、表示手段にてデバッグ結果を可視化するための情報を簡易に設定・変更などできるので、並列プログラム内に可視化のための関数などをプログラムに埋め込むなどして並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる。

【0027】

【実施例】以下、図面を参照しながら本発明の実施例を説明する。

(1) 最初に、並列プログラムデバッガの実施例について説明する。

(第1の実施例) 図1に、本実施例の並列プログラムデバッガの構成を示す。図のように、この並列プログラムデバッガは、デバッガコントローラ2と複数の逐次デバッガ4を備える。逐次デバッガ4の個数は任意であるが、本実施例では5個の逐次デバッガ4-1~4-5を備えているものとして説明をする。

【0028】逐次デバッガ4は、逐次プログラムのデバッグを行うためのデバッガである。本実施例では、逐次デバッガ4として、例えばunix上で動作するdbx

やgdbのようなデバッガを想定することとする。

【0029】各々の逐次デバッガ4は、デバッガコントローラ2により制御される。なお、本実施例では、発明を理解しやすくするために、逐次デバッガ4を用いてデバッグを施される複数のユーザプログラム6（6-1～6-5）はデータパラレルプログラムつまり全ての部分プログラムが同一のプログラムであるものとしているが、もちろん個々のプログラムが互いに異なる場合でも本発明は適用可能である。

【0030】本実施例のデバッガコントローラ2は、命令解釈実行部21、第1の通信部22-1、第2の通信部22-2、ファイル入出力部23、実行制御命令記憶部24、コマンド入出力部25を備える。

【0031】実行制御命令記憶部24は、並列プログラムの実行制御命令記述ファイル3を記憶する。実行制御命令記述ファイル3は、並列プログラムの実行制御命令を記述するためのファイルである。詳しくは後述するが、本実施例では、例えば停止したプログラムの番号と各プログラムの停止位置（ブレークポイントの位置）とブレークポイントでの停止時のプログラム内で使われている変数の値と当該デバッガコントローラ2内で保持されている内部変数（ローカル変数）の値を用いて記述される。

【0032】ファイル入出力部23は、外部ファイルである実行制御命令記述ファイル3の入出力を行うための部分である。命令解釈実行部21は、実行制御命令記述ファイル3の内容を解釈して並列プログラムを制御するためのデバッグコマンドを逐次デバッガ4に対して発行する。また、命令解釈実行部21は、コマンド入出力部25、通信部22-1、通信部22-2、ファイル入出力部23および実行制御命令記憶部24との間で通信を行いながら、デバッグ処理を進めて行く。

【0033】通信部22-1は、個々の逐次デバッガ4と命令解釈実行部21の間の通信を補助するためのものである。通信部22-2は、外部プロセス10と命令解釈実行部21の間の通信を補助するためのものである。

【0034】コマンド入出力部25は、ユーザ（図中の8）との入出力インタフェースである。なお、図2に、本実施例のデバッグコントローラ2における処理のフローチャートを示す。ただし、ステップS6内の各命令はすべてが必須ではなく、後述するように必要に応じて適宜容易されるものである。

【0035】以下、本実施例の並列プログラムのデバッグ処理について説明する。本実施例の並列プログラムデバッガでは、まず、ユーザは、デバッガコントローラ2を起動し、続いて逐次デバッガ4上からユーザプログラム6を起動する。このとき、逐次デバッガ4の入出力は、デバッガコントローラ2の通信部22-1と接続する。

【0036】デバッガコントローラ2の命令解釈実行部

21は、ユーザからのコマンドを待ち、コマンドが入力されたら、そのコマンドに応じた処理を行い、再びユーザのコマンドを待つ。

【0037】本実施例の並列プログラムデバッガでは、条件付き実行モードと対話モードを切り換えて使用できる。条件付き実行モードにおいては、条件付き実行命令と条件付き実行継続命令を備えている。これら条件付き実行命令あるいは条件付き実行継続命令が実行された場合は、命令解釈実行部21は、実行制御命令記憶部24に記憶されている命令文を解釈しながら、通信部22-1を通じて個々の逐次デバッガ4（4-1～4-5）にデバッグコマンドを送る処理や、デバッガコントローラ2内部に内部変数を作成する処理や、通信部22-2を通じて外部プロセス10にデータを送る処理を行う。

【0038】また、本実施例の並列プログラムデバッガは、対話モードにおいては、対話型の実行を行うための対話型実行命令と対話型実行継続命令を備えている。これら対話型実行命令あるいは対話型実行継続命令が実行された場合には、命令解釈実行部21は、デバッガコントローラ2内部のデータ構造や変数の値に応じて通信部22-1を通じて個々の逐次デバッガ（4-1～4-5）にデバッグコマンドを送る処理や、デバッガコントローラ2内部に内部変数を作成する処理や、通信部22-2を通じて外部プロセス10にデータを送る処理を行う。

【0039】次に、逐次デバッガ4の機能について説明する。本実施例では、各逐次デバッガ4は、以下に列挙する機能を持つとする。「runコマンド」は、逐次プログラム6を先頭から実行させる。

【0040】「contコマンド」は、一時停止している逐次プログラム6の実行を再開させる。「print x コマンド」は、一時停止している逐次プログラム6の内部変数を出力する。

【0041】「stop at line コマンド」（lineは行番号を表す値である；以下、同様である）は、逐次プログラム6のline行にブレークポイントを設定する。ブレークポイント停止時には、停止した行の行番号が出力される。

【0042】「ctrl-C コマンド」は、実行中の逐次プログラム6を強制的に一時停止させる。次に、デバッガコントローラ2の機能について説明する。

【0043】デバッガコントローラ2内部に条件付き実行モードと対話モードの2つのモードを識別するフラグを設ける。デバッガコントローラ2の起動後のモードは、対話モードである。

【0044】対話モードでは、ユーザからの命令は通信部22-1を通じて各逐次デバッガ4に伝える。デバッガコントローラ2は、例えば次に示すような命令がユーザから入力されるのを待つ。

【0045】「select 命令」は、ユーザからの

命令を伝える逐次デバッガ4を指定する。起動後は、全てのプログラムが指定されている「stop at line 命令」は、line行でブレークポイントを設定する命令である。この命令を用いてブレークポイントを設定する毎に、デバッガコントローラ2側で一意的ブレークポイント番号をつける。この番号と、行番号の対応表（以下、行-ブレークポイント対応表と呼ぶ）を作成し、デバッガコントローラ2で保持する。

【0046】「readcondfile 命令」は、外部からファイルを実行制御命令記憶部24に読み込む。「crun 命令」は、モードを条件付き実行モードに移し、全ての逐次デバッガ4に「run 命令」を送り、全てのプログラム6がブレークポイントで停止するか、または終了するのを待ち、後述する変数設定処理を行い、“\$CONTPELIST”という変数に“#PEALL”をセットし、実行制御命令記憶部24内に保持されている実行制御命令記述プログラムを解釈し実行する。ここで、#PEALLは、すべてのプログラムを示す変数とする。全てのプログラムが終了した場合には、対話モードに戻る。

【0047】「ccont 命令」は、\$CONTPELISTに登録されているプログラム6を制御する逐次デバッガ4に「cont 命令」を送り、全てのプログラムがブレークポイントで停止するか、または終了するのを待ち、後述する変数設定処理を行い、\$CONTPELISTを#PEALLにし、実行制御命令記憶部24内に保持されている命令を解釈し実行する。全てのプログラムが終了した場合には、対話モードに戻る。

【0048】「view variable コマンド」は、実行制御命令記述ファイル内の変数“variable”を通信部22-2を通じて外部のプロセス10へと送る。

【0049】次に、実行制御命令記述ファイル3の記述方法について説明する。本実施例では、実行制御命令記述ファイル3は、例えばC言語形式のようなプログラムで記述する。

【0050】まず、実行制御命令記述ファイル3で用いられる変数とその設定方法について示す。以下、\$varは、内部変数を表すものとする。“#PE”はプログラムの番号、“#PE:var”は#PE番のプログラム中の変数の値、“#BP(#PE)”は#PE番のプログラムが停止しているブレークポイントの位置とする。例えば、1番のプログラムが2番のブレークポイントで停止した場合は、#BP(1)=2となる。

【0051】変数設定処理は、すべてのプログラムについて#BP(#PE)の値を設定する。この処理は、逐次プログラム6がブレークポイントで停止した時に、その逐次プログラム6を制御する逐次デバッガ4が通信部22-1に対して出力する停止したブレークポイントの行番号から、行-ブレークポイント番号対応表を用いて

ブレークポイントの番号を調べ、その値を#BP(#PE)の値とすることで実現される。また、変数設定処理では、“\$PENUM”の値がプログラムの数に設定される。

【0052】実行制御命令記述ファイル3内で#PE:varが参照された場合は、#PE番の逐次プログラム6を制御する逐次デバッガ4に対して、「print var 命令」を送り、その結果を#PE:varとして用いる。

【0053】次に、実行制御命令記述ファイル3で用いられる関数とその実現方法について示す。関数BP(#PEi, #PEj, ..., #PEk)は、引数に与えられたプログラムが停止しているBPの位置の種類を返す。ここで、 $1 \leq \#PEi, \#PEj, \#PEk \leq n$ であり、nは逐次プログラムの数とする。これは、#BP(#PE)の値をすべてのプログラム番号について調べ、異なる種類のものを全て列挙することで求める。

【0054】関数PE(#BP1, #BP2, ..., #BPm)は、引数に与えられたBPで停止しているプログラムのリストを返す。これは、#BP(#PE)の値を $1 \leq \#PE \leq n$ について調べ、その結果が引数に含まれる場合には、#PEの値を結果として返すことによって求める。

【0055】関数CONT(#PEi, #PEj, ..., #PEk)は、引数に与えられた逐次プログラムに対して逐次デバッガを通して「cont 命令」を送る。関数WAIT(#PEi, #PEj, ..., #PEk)は、引数に与えられた全てのプログラムのデバッガからメッセージが返って来るのを待つ。全てのプログラムのデバッガからメッセージが返って来た後、変数設定処理を行う。もし、全てのプログラムが終了していた場合には、「crun 命令」あるいは、「ccont 命令」を終了し、デバッガコントローラのモードを対話モードに戻す。

【0056】さて以下では、上記したような命令などの各機能を用いて、ある停止条件でプログラムを停止させる手順を示す。ここでは、停止条件の一例として、次のような条件を取り上げる。図3のプログラムが5つ同時に動作する場合において、02行目と04行目にブレークポイントを設定し個々のプログラムを実行させると、個々のプログラムはwhileループを一回りする毎にどちらかのブレークポイントで停止する。停止条件は、この「while ループ」1回毎の停止を繰り返した際に、ブレークポイント1で停止しているプログラムが2つだけある場合以外にはプログラムを停止させずに、ブレークポイント1で停止しているプログラムが2つだけある場合とする。このような停止条件が成立した場合にはじめてプログラムを停止させるという処理を実現する手続きについて説明する。

【0057】まず、ユーザはデバッガコントローラ2を

起動し、続いて逐次デバッガ4上からユーザプログラム6を起動する。続いて、「select (#PEALL) 命令」ですべてのプログラムを選択した後、「stop at 02 命令」と「stop at 04 命令」で、個々のプログラムの02行目と04行目にブレークポイントを設定する。この時、02行目のブレークポイントには1のブレークポイント番号が、04行目のブレークポイントには2のブレークポイント番号が夫々付けられたものとする。

【0058】ユーザは、図4に示すような実行制御命令を記述した実行制御命令記述ファイル3を作成する（あるいは既に作成してあるものとする）。続いて、「readcondfile 命令」で、このファイルをデバッガコントローラ2内の実行制御命令記憶部24に読み込む。

【0059】続いて、「crun 命令」でプログラム6を条件付き実行する。図3に示すような個々のプログラム6は、それぞれyの値に応じて1番か2番のブレークポイントで停止し、デバッガコントローラ2に、何行目で停止したかというメッセージを返す。全てのプログラムから、ブレークポイントに達したというメッセージあるいはプログラムが終了したというメッセージが届いた後、実行制御命令文の解釈実行を始める。

【0060】図4の実行制御命令文において、\$BP1NUMはブレークポイント1で停止しているプログラムの個数を数える変数であり、「for ループ」が実際に数をカウントしている。個数が2である場合は「break 命令」で処理をユーザに戻し、そうでない場合はWAIT (#PEALL) で再び全てのプログラムを継続実行させ、再びブレークポイントでの停止条件を判定する。

【0061】以上の手順で、ユーザは所望の停止位置でプログラムを停止させることができる。この処理の後、ユーザは、「ccont 命令」で操作を続けられ、再び所望の条件でプログラムを停止させることができる。

【0062】このように、複数の逐次プログラム間である条件が成立した場合に、自動的にプログラムを停止させるようなデバッグ処理が可能となる。もちろん、本発明は、上記のようなデバッグ処理だけでなく、複数の逐次プログラム相互間の動作状態を関連付けた他の所望の内容のデバッグ処理を行うことができる。

【0063】ところで、上述した命令以外に適宜命令を容易することにより、所望のデバッグ処理を実現することができる。以下では、種々のデバッグ処理の例を示す。まず、ユーザコマンドにより、デバッガコントローラ2内にプログラムの停止位置およびその停止位置で参照したい変数を構造体として持ち、並列プログラム実行の際に制御条件として参照する例について説明する。

【0064】プログラムの別の場所にある変数を他のブレークポイントで参照したい場合には、「probe

命令」をさらに設ければ良い。「probe at line x \$x 命令」は、ブローポイント設定のための命令であり、対話モードおよび条件付き実行モードの両方で使用できるものとする。

【0065】この命令により、並列プログラムデバッガ内部に、例えば以下に示すような構造体PROBEが用意され、構造体内のlineに引数line、構造体内のvar1に引数x、構造体内のvar2に引数\$xが設定され、逐次デバッガ4を通じてline行にブレークポイントが設定される。

```
struct PROBE {
    int      line ;    /*行番号*/
    char[100] var1 ;    /*プログラム変数*/
    char[100] var2 ;    /*ローカル変数*/
}
```

デバッガコントローラ2に用意される「ucrun 命令」は、対話モードでのプログラム実行命令である。この命令は、モードを対話モードに切替え、全ての逐次デバッガ4に「run 命令」を送り、\$CONTPELISTに#PEALLを代入し、全てのプログラム6がブレークポイントで停止するかまたは終了するのを待つ。あるプログラム#PEの停止位置がブローポイントの場合は、デバッガコントローラ2がそのプログラム#PEに対して「print x 命令」を発行し、その結果を変数#PE:\$xに代入し、再びプログラム#PEに対して「cont 命令」を与える。

【0066】デバッガコントローラ2に用意される「uccont 命令」は、対話モードでのプログラム継続実行命令である。この命令は、\$CONTPELISTに代入されているプログラム番号のプログラムを制御している逐次デバッガ4に「cont 命令」を送り、\$CONTPELISTに#PEALLを代入し、全てのプログラム6がブレークポイントで停止するか、または終了するのを待つ。あるプログラム#PEの停止位置がprobeポイントの場合は、デバッガコントローラ2がそのプログラム#PEに対して「print x 命令」を発行し、その結果を変数#PE:\$xに代入し、再びプログラム#PEに対して「cont 命令」を与える。

【0067】図5はある行で計算された値が後の行の命令によって破壊されてしまうプログラムの例である。このプログラムでは、01行目で計算されたxの値が03行目の命令によって破壊され04行目で参照することはできない。

【0068】ここでは、上記プログラムにおいて、ユーザが04行目でブレークポイントを設定し、そこでプログラムが停止した際に、01行目で計算されたxの値を参照するという場合に、02行目に対して「probe 命令」を適用する例を説明する。

【0069】まず、ユーザはデバッガコントローラ2を

起動し、続いて逐次デバッガ4上からユーザプログラム6を起動する。続いて、「select (#PEALL) 命令」ですべてのプログラム6を選択した後、「probe at 02 x \$x 命令」で02行目をブローポイントとし、「stop at 04 命令」でブレークポイントを設定する。02行目のPはブローポイントが、04行目のBはブレークポイントがそれぞれ設定されたことを示す。

【0070】「ucrun 命令」でプログラムが実行されると、まず02行目で各プログラムは停止し、xの値が読み出され、それぞれ、#PE:\$xの値に格納される。プログラムは再実行され、各々のプログラムは04のブレークポイントで停止し、ユーザに入力要求が戻る。この時、変数#PE:\$xに所望の変数の値が格納される。

【0071】このように、プログラムのある時点での変数の値を、そのプログラムの変数の値がプログラムの継続実行によって破壊された後でも参照することができる。次に、ユーザコマンドにより、デバッグコントローラ2内にブレークポイントの階層関係を構造体として持ち、並列プログラム実行の際に実行制御条件として参照する例について説明する。

【0072】ここでは、図6に示すプログラムで、ブレークポイントをそれぞれ、04、06、09、11行目に設定し、収束ループ3が終わるまでは、収束ループ2の処理を進めず、収束ループ2が終わるまでは、収束ループ1の処理を進めないという処理を行う場合の例を示す。なお、04、06、09、11行目のブレークポイ

```
struct BP{
    int      BPnum; /*ブレークポイント番号*/
    struct *BP child; /*子供のブレークポイント番号*/
    struct *BP group; /*子供のブレークポイント番号*/
}
```

この命令により、bp1をBPnumとする構造体のchildはbp2をBPnumに持つ構造体を指すこととなる。

【0078】「group bp1 bp2 命令」は、ブレークポイントbp1とbp2の間にグループ関係を定義する。データ構造BPが2つ用意され、一方の構造体のBPnumには、bp1のブレークポイント番号が、他方の構造体のBPnumには、bp2のブレークポイント番号が代入される。この命令により、bp1をBPnumとする構造体のgroupは、bp2をBPnumに持つ構造体を指すこととなる。

【0079】デバッガコントローラ2に用意される「ucrun 命令」は、対話モードでのプログラム実行命令である。この命令は、モードを対話モードに切替え、全ての逐次デバッガに「run 命令」を送り、\$CONTPELISTに#PEALLを代入し、全てのプログラムがブレークポイントで停止するか、プログラムが

ントは、それぞれ、ブレークポイント番号1、2、3、4に対応するものとする。

【0073】本実施例において、ユーザの最初の「ucrun 命令」とそれ以降の「uccont 命令」の繰り返しによって停止するブレークポイントの位置を図7に示す。

【0074】1回目では、個々のプログラムはそれぞれ2、1、1、1、1のブレークポイントで停止する。2回目では、個々のプログラムはそれぞれ1、2、1、2、1のブレークポイントで停止する。

【0075】3回目では、2番と3番のプログラムが3のブレークポイントで停止されるので、2番と3番のプログラムは、他のプログラムが収束ループ3を終了するまで継続実行をされない。

【0076】5回目で全てのプログラムが3のブレークポイントに到達するので、この時全てのプログラムは継続実行され、6回目で、個々のプログラムは、それぞれ1、4、2、1、1のブレークポイントで停止することになる。

【0077】この処理は、次に示す「parent 命令」および「gourp 命令」をさらに設けることで実現できる。「parent bp1 bp2 命令」は、ブレークポイントbp1とbp2の間に親子関係を定義する。以下に示すようなデータ構造BPが2つ用意され、一方の構造体のBPnumには、bp1のブレークポイント番号が、他方の構造体のBPnumには、bp2のブレークポイント番号が代入される。

終了するのを待つ。ブレークポイントに親子関係が設定されている場合は、返ってきたブレークポイントの種類が一番子供であるブレークポイント番号で停止しているプログラム番号を内部変数\$CONTPELISTに代入する。

【0080】デバッガコントローラ2に用意される「uccont 命令」は、対話モードでのプログラム継続実行命令である。「uccont 命令」は、\$CONTPELISTに代入されているプログラム番号のプログラムを制御している逐次デバッガに「cont 命令」を送り、\$CONTPELISTに#PEALLを代入し、全てのプログラムがブレークポイントで停止するか、プログラムが終了するのを待つ。ブレークポイントに親子関係が設定されている場合は、返ってきたブレークポイントの種類が一番子供であるブレークポイント番号で停止しているプログラム番号を内部変数\$CONTPELISTに代入する。

【0081】本実施例の場合、まず、ユーザはデバッグコントローラ2を起動し、続いて逐次デバッグ上からユーザプログラムを起動する。続いて、「select (#PEALL) 命令」ですべてのプログラムを選択した後、「stop at 04, stop at 06, stop at 09, stop at 11 命令」で、それぞれ1, 2, 3, 4番のブレークポイント番号のブレークポイントを設定する。

【0082】続いて、「group 1, 2, parent 3, 1, parent 4, 3 命令」を実行すると、例えば図8に示すような連結リストがデバッグコントローラ内部に作成される。連結リスト(図中、30, 31, 32, 33)は、ブレークポイント番号(図中、30-1)、子へのリンク(図中、30-2)、グループへのリンク(図中、30-3)からなる。子へのリンクやグループへのリンクが存在しないものは、図中の33-2, 33-3のようにnullがセットされる。

【0083】「ucrun 命令」を実行すると、\$CONTPELISTに#PEALLが設定され、各プログラムが停止した位置がデバッグコントローラ2に返される。この時のブレークポイントの種類を調べ、「group 命令」および「parent 命令」で作成された構造体の連結リストを参照しながら、一番子供のブレークポイントの種類を決定する。このブレークポイントで停止しているプログラムの番号が、内部変数\$CONTPELISTに代入される。

【0084】「uccount 命令」を実行すると、\$CONTPELISTに代入されているプログラム番号のプログラムにcontが送られる。続いて、\$CONTPELISTに#PEALLが設定され、各プログラムの停止後に各プログラムが停止した位置がデバッグコントローラ2に返される。この時のブレークポイントの種類を調べ、「group 命令」および「parent 命令」で作成された構造体の連結リストを参照しながら、一番子供のブレークポイントの種類を決定する。このブレークポイントで停止しているプログラムの番号が、内部変数\$CONTPELISTに代入される。

【0085】このように、ブレークポイントに階層性を持たせることで、並列プログラムの実行制御条件を簡便に設定できる。次に、デバッグコントローラに対して、処理の中断命令を備え、中断時の個々のプログラムの停止状況を報告させる例について説明する。

【0086】ユーザは、「ctrl-C、命令」をデバッグコントローラ2に送ることにより、コントロールをユーザへと戻すことができる。「ctrl-C 命令」は、実行中のプログラム6の逐次デバッグ4が、そのプログラム6に対して逐次デバッグ4を通じて「ctrl-C コマンド」を送るように命令し、その時のプログラム6の停止位置を受取りユーザに示す。

【0087】これによって、例えばブレークポイントと個々のプログラムそのものに埋め込まれた通信命令によってデッドロックが生じた場合、デバッグコントローラ2の処理を中断し、プログラムの停止状態をユーザに通知し、ユーザに対するコマンドの受け付けを再開することができる。

【0088】このように、誤ったブレークポイントの設定などによって、プログラムの制御がデッドロックを起こした場合に、ユーザが割込みをかけ、デッドロックが起こった状況を把握することができる。

【0089】(2)次に、並列プログラムを構成する個々の部分プログラムを逐次デバッグ上から起動し、逐次デバッグを通して個々の部分プログラムからプログラム実行時の変数の値を取得し可視化を行う並列プログラム可視化装置の実施例について説明する。

【0090】(第2の実施例)図9に、本実施例の並列プログラム可視化装置の構成を示す。図のように、この並列プログラム可視化装置は、デバッグコントローラ102、複数の逐次デバッグ104(104-1~104-n)、1台または複数台の表示部107(107-1~107-m)を備える。

【0091】逐次デバッグ104は、逐次プログラム106のデバッグを行うためのデバッグである。本実施例では、逐次デバッグ104として、例えばunix上で動作するdbxやgdbのようなデバッグを想定することとする。

【0092】各々の逐次デバッグ104は、デバッグコントローラ102により制御される。なお、本実施例では、発明を理解しやすくするために、逐次デバッグ4を用いてデバッグを施される複数のユーザプログラム6(6-1~6-5)はデータパラレルプログラムあるいは全ての部分プログラムが同一のプログラムであるものとしているが、もちろん個々のプログラムが互いに異なる場合でも本発明は適用可能である。なお、本実施例は、プログラムが1つしかない場合でも適用可能である。

【0093】デバッグコントローラ2は、概略的には、並列プログラム6を構成する個々の部分プログラムをデバッグする逐次デバッグ4に対してデバッグコントローラ2は命令を送り、逐次デバッグ4からの出力を受け取るものである。

【0094】本実施例のデバッグコントローラ102は、制御部121、第1の通信部122-1、第2の通信部122-2、ユーザインタフェース125、変数取得表131、変数記憶表132を備える。

【0095】制御部121は、第1の通信部122-1、第2の通信部122-2、ユーザインタフェース125、変数取得表131および変数記憶表132との間で通信を行いながら、本システム全体を制御していく。詳しくは後述するが、概略的には、並列プログラム実行

後、プログラムの進行に合わせて変数取得表131を参照しながら、各々の逐次デバッガ4に対して命令を送りあるいは変数値を受け取るなどの制御を行う。

【0096】通信部122-1は、個々の逐次デバッガ104との通信を行う。通信部122-2は、各表示部107との通信を行う。変数取得表131は、変数の取得位置ならびに取得する変数に関する情報を記憶する。例えば、後述するようにユーザにより入力された、ブローポイント、ブロー変数、ショウポイントなどを格納する。

【0097】変数記憶表132は、取得した変数の値を記憶する。ユーザインターフェース部125は、ユーザ（図中の108）との入出力インターフェースである。

【0098】デバッガコントローラ2には、1つまたは複数の表示部107（107-1～107-m）が用意される。表示部107は、一つ以上の変数（テンプレート変数と呼ぶ）の値をもとに可視化を行うための装置である。例えば、プログラム6がショウポイントに到達すると、ユーザがあらかじめ選んだ表示部107で可視化が行われる。

【0099】各表示部107は、通信部140と可視化処理部141を備えている。通信部140は、デバッガコントローラ2との通信を行う。可視化処理部141は、テンプレート変数記述部142、式解釈部143、描画部144を有する。

【0100】テンプレート変数記述部142は、各々のテンプレート変数に対応するプログラム中の変数や、それらを組み合わせた式を記述する。例えば、ブロー変数、プログラム番号、ブレークポイント番号やこれらを組み合わせた式を記述できる。

【0101】表示部107では、テンプレート変数記述部142に記述された記述が式解釈部143で解釈され、描画部144が描画を行う。次に、本実施例の並列プログラム可視化装置の動作の概略を示す。

【0102】まず、ユーザはデバッガコントローラ102を起動し、続いて逐次デバッガ104-1～104-n上からユーザプログラム106-1～106-nを起動する。このとき、逐次デバッガ104の入出力はデバッガコントローラ102の通信部122-1と接続する。

【0103】ユーザは、ユーザインターフェース部125を通じて、変数の取得位置（ブローポイント）、変数名（ブロー変数）、可視化を行う位置（ショウポイント）を入力し、制御部121はこれらの情報を変数取得表131に格納する。

【0104】制御部121は、変数取得表の内容に基づいて個々の逐次プログラム106-1～106-nに対してブレークポイントを設定する。並列プログラム実行後、制御部121は変数取得表131の値を参照しながら、個々のプログラム106が停止した位置に応じて、

個々の逐次デバッガ104に継続命令や変数取得のための命令を送る。取得された変数は、変数記憶表132に格納される。

【0105】ユーザの表示部選択命令により、使用する表示部107が選択される。表示部107が選択されると、テンプレート変数記述部142に対する入力要求をユーザに対して行う。プログラムがショウポイントに到達した際には、選択されている表示部107を用いて可視化が行われる。

【0106】表示部107では、デバッガコントローラ102の通信部122-2から表示部107の通信部140を通じて、必要な変数の値を受取り、テンプレート変数記述部142に記述された記述を式解釈部143で解釈しながら、描画部144が描画を行う。

【0107】以下、本実施例についてより詳しく説明する。デバッガコントローラ102は、ユーザに対して、可視化のために用いる表示部107を選択するための「view 命令」と、ブロー変数やブローポイントを指定するための「probe 命令」と、ショウポイントを指定するための「show at 命令」を有する。また、デバッガコントローラ102は、「run 命令」と「cont 命令」を持つ。「run 命令」は、逐次デバッガ104に伝えられプログラム106が実行される。「cont 命令」は、逐次デバッガ104に伝えられプログラム106が継続実行される。

【0108】「view 命令」は、引数として表示部107の番号をとる。表示部107には予め番号が付けられているものとする。「view 命令」を実行すると可視化のために必要な変数に関する情報を入力するウィンドウが開く。

【0109】図10は、2次元の点を表示する表示部107の一例である。点の位置は（X，Y）で表される。このX，Yをテンプレート変数と呼ぶ。それぞれのテンプレート変数とユーザプログラム中のどの変数が対応するのかを入力する。図10のX＝，Y＝の覧にそれぞれx，y（図中、185，186）と書いてあるのは、プログラム中からx，yという変数の値を取得し、2次元のX，Y軸上にプロットすることを示している。縦方向と横方向にはそれぞれスクロールバー（図中、201，200）が付いており、画面のスクロールができる。XmaxとYmaxの覧にそれぞれ20，20（図中、183，184）が入力されているのは画面上の表示の倍率を設定するものである。この入力の代わりに、拡大、縮小ボタンを設けてもよい。

【0110】「probe 命令」は、引数として行番号と変数の列をとる。この命令が実行されると、引数として与えられた行番号にブレークポイントが設定される。この行番号に相当する行をブローポイントと呼ぶ。取得する変数のリストを変数取得表131に、行番号、ブローポイントを示すフラグP、変数リストとい

うフォーマットで登録する。例えば、10行目でxとyという変数を取得するという「probe 10 x y 命令」が実行された場合には、図11に示すような要素が変数取得表131に登録される。

【0111】プログラム106-1~106-nを実行し、プログラムがブローポイントに到達した際には、デバッガコントローラ102は、各々の逐次デバッグ104-1~104-nに対して変数取得表131に登録してある変数の値を取得する命令を発行する。具体的には、xという変数の値を取得する場合には、逐次デバッグ104に対して、「print x コマンド」を発行する。デバッガコントローラ102は、この結果を変数記憶表132に格納する。このとき、格納する変数には、変数を取得したプログラムの番号と、ブローポイントの行番号を付けておく。例えば、10行のブローポイントでプロセッサ番号0から4番のプログラムからxとyという変数の値を取得した場合、図12に示すような変数記憶表132が作成される。この後、デバッガコントローラ102は、個々の逐次デバッグ104-1~104-nに「cont 命令」を送り、プログラムを継続実行する。

【0112】「show at 命令」は、引数として行番号をとる。この命令が実行されると、引数として与えられた行番号にブレークポイントが設定される。この行番号に相当する行をショウポイントと呼ぶことにする。行番号、ショウポイントを示すフラグSが変数取得表に格納される。例えば、11行をショウポイントとする「show at 11 命令」が実行された場合には、図13に示すような値が変数取得表131に登録される。

【0113】デバッガコントローラ102は、ユーザが「run 命令」あるいは「cont 命令」を実行した後に、全ての部分プログラムがショウポイントに到達するか、またはプログラムが終了するのを待つ。

【0114】すべての部分プログラムが、ショウポイントに到達するかまたは終了した場合、ユーザがあらかじめ選択した表示部107に視覚化のためのデータを渡し、視覚化を行うように命令する。

【0115】同一行に「probe 命令」と「show at 命令」が設定された場合は、「probe 命令」に対して行われる処理が優先され、プログラムを継続実行せずに「show at 命令」に対して行われる処理が実行される。

【0116】今、「view 命令」で二次元表示を行う表示部107を指定し、そのテンプレート変数X、Yをそれぞれ、x、yとしXmax、Ymaxをそれぞれ20とし、「probe 命令」で、ある並列プログラム（ただし、並列プログラムを構成する全ての部分プログラムは同じものとする）の10行目ブローポイントとしx、yを取得するように設定し、11行目にショウ

ポイントを設定して、「run 命令」でプログラムを実行させると、プログラムは11行目で停止し、図12のような表が得られる。その変数の内容は、表示部107へと送られる。さらに、デバッガコントローラ102は、表示部107に可視化を行うように命令を送る。表示部107は、このデータと命令を受け、テンプレート変数記述部142に記述された内容を解釈しながら図14のような表示を描画部144を用いて行う。

【0117】なお、表示部107として3次元表示を行うものを容易し、これを選択すれば3次元表示を行うことも可能である。このように本実施例によれば、デバッグを通じて変数を取得し、表示部で描画を行うことで、デバッグ対象となるプログラムには変更を加えることなく、プログラムの挙動を可視化することができる。また、テンプレート変数記述部にブロー変数、プログラム番号、ブレークポイント番号やそれらを組み合わせた式を記入できるので、幅広い視点からの可視化が可能である。

【0118】ここで、テンプレート変数として、PE番号、ブレークポイント番号を書けるようにしても良い。#PE、#BP（図中、187、188）はそれぞれ、プログラム番号、ブレークポイント番号を示す変数とする。これにより、図15に示すように、どのプログラムがどのブレークポイントで停止しているかを表示させることができる。

【0119】また、テンプレート変数の対応の記述の際に、右辺に式を書けるようにしても良い。この場合、デバッガコントローラ102内に、前述したような式解釈部を設け、その式の値を計算できるようにする。これにより、図16に示すように、X軸をxとし、Y軸をx+yとしたようなグラフを書くことができる。

【0120】このように本実施例によれば、並列プログラム内に可視化のための関数などをプログラムに埋め込むなどして並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる。

【0121】（第3の実施例）図17に、本実施例の並列プログラム可視化装置の構成を示す。本実施例は、第2の実施例の構成において、デバッガコントローラ102内部に簡易言語処理部160を付加したものである。

【0122】この簡易言語処理部160は、簡易言語記憶部161、簡易言語解釈実行部162、簡易言語変数記憶部163からなる。簡易言語記憶部161は、個々の部分プログラムが特定行に到着したときに実行される簡易言語を記述しておくためのものである。簡易言語解釈実行部162は、簡易言語を解釈して実行するためのものである。簡易言語変数記憶部163は、簡易言語実行時に用いられた変数の値を記憶しておくためのものである。

【0123】本実施例の並列プログラム可視化装置の動作の概略を以下に示す。まず、ユーザはデバッガコント

ローラ102を起動し、続いて逐次デバッグ104-1~104-n上からユーザプログラム106-1~106-nを起動する。このとき、逐次デバッグ104の入出力はデバッグコントローラ102の通信部122-1と接続する。

【0124】ユーザは、ユーザインターフェース部125を通じて、変数の取得位置（プローブポイント）、変数名（プローブ変数）、可視化を行う位置（ショウポイント）、簡易言語を実行する位置（簡易言語実行ポイント）を入力し、制御部121はこれらの情報を変数取得表131に格納する。

【0125】制御部121は、変数取得表の内容に基づいて個々の逐次プログラム106-1~106-nに対してブレークポイントを設定する。並列プログラム実行後、制御部121は変数取得表131の値を参照しながら、個々のプログラム106が停止した位置に応じて、個々の逐次デバッグ104に継続命令や変数取得のための命令を送る。取得された変数は、変数記憶表132に格納される。

【0126】プログラムが簡易言語実行位置に到達した場合は簡易言語の実行を行う。簡易言語は、簡易言語処理部160にて解釈、実行される。ユーザの表示部選択命令により、使用する表示部107が選択される。表示部107が選択されると、テンプレート変数記述部142に対する入力要求をユーザに対して行う。プログラムがショウポイントに到達した際には、選択されている表示部107を用いて可視化が行われる。

【0127】表示部107では、デバッグコントローラ102の通信部122-2から表示部107の通信部140を通じて、必要な変数の値を受取り、テンプレート変数記述部142に記述された記述を式解釈部143で解釈しながら、描画部144が描画を行う。

【0128】以下、本実施例についてより詳しく説明する。本実施例では、ユーザが簡易言語を記述することができる。この簡易言語は、外部変数として、「probe 命令」で取得した変数を利用できるものとする。簡易言語を起動する位置を決める「user 命令」を用意する。「probe 命令」は「user 命令」に優先し、「user 命令」は「show at 命令」に優先する。

【0129】簡易言語中で使用される「show 命令」は、簡易言語の中から表示部107に対して、可視化のために必要なデータを与えると同時に、表示部107に表示を行うよう指示する命令である。

【0130】例えば、簡易言語として、図18のように記述したものとする。この例では、01行目でプローブ変数x、yを使うことを宣言し、02で2番の可視化ツールで可視化を行っている。

【0131】2番の可視化ツールが二次元描画のものであり、 $X = oldx - x$ 、 $Y = oldy - y$ とすると、

図19のようにx、yが移動した方向を表示できる。このように本実施例によれば、ユーザプログラムでは既に更新されてしまっていることのできない変数の過去の値を利用した柔軟な可視化が可能になる。

【0132】なお、簡易言語および簡易言語起動位置は、複数あっても良い。

（第4の実施例）図20に、本実施例の並列プログラム可視化装置の表示部207の構成を示す。本実施例は、第1の実施例または第2の実施例の構成において、表示部107内部に、ユーザイベントハンドラ210、図形データ記憶部211、プロセッサマップ表示部212を付加したものである。

【0133】図形データ記憶部211は、可視化された画面上の図形の座標と、その図形を描画するためのデータを取得したプログラムの番号とプログラム中の位置を記憶する。プロセッサマップ表示部212は、プログラムの番号を2次元の表に並べたプロセッサマップを表示する。

【0134】本実施例では、ユーザが可視化された画面上の特定の図形データをマウスでクリックした場合、ユーザイベントハンドラ210がマウスのクリックされた位置と図形データ記憶部211の内容から、ユーザがクリックした図形を描画するためのデータを取得したプログラムの番号を取得し、プロセッサマップ上のそのプログラム番号の表示を反転させる。

【0135】二次元の表示を行う表示部107で点を描画する際に、画面上の点の座標すなわち二次元表示における点の相対座標（x、y）、これら変数x、yを取得したプロセッサの番号、プローブポイントの位置、ショウポイントの位置を互いに関連付けて記憶しておく。

【0136】図14は、5つの点が描かれている画面であり、それぞれの点に対して、図形データ記憶部211に図21のようなデータが保持する。図形番号は表示された図形にユニークに与えられる番号であり、mouse X、mouse Yは画面上の位置であり、x、yは表示部107による二次元表示における相対座標であり、プロセッサ番号は、その変数が計算されたプロセッサの番号であり、ショウポイントは表示が行われたショウポイントを示している。

【0137】ユーザが画面上の点をクリックしたとき、ユーザイベントハンドラ210は、マウスのクリックされた座標を検出し、図形データ記憶部211のmouse X、mouse Yの値と一致するものを探し、その図形番号を得る。その図形番号のプロセッサ番号を表から取得し、そのプログラム番号をプロセッサマップ表示部212に送る。プロセッサマップ表示部212は、例えば図22のように、送られた番号のプロセッサ番号を反転する。

【0138】このようにすれば、可視化画面から、ユーザにとって好適な情報を得ることができうる。なお、ブ

ロセッサマップ表示部 212 の代わりにソースファイルを表示するプロセスを設け、図形データ記憶部 211 の変数リストに、その変数が取得されたブローポイント記憶することで、表示された点をマウスでクリックした場合に、その点を描くためのデータが取得されたブローポイントの行を反転するという機能が得られる。また、本発明は上述した各実施例に限定されるものではなく、その要旨を逸脱しない範囲で、種々変形して実施することができる。

【0139】

【発明の効果】

(1) 本発明では、並列プログラムに対する所望のデバッグ内容の記述を外部からデバッガコントローラに与え、デバッガコントローラではこの記述の解釈結果に従い各デバッガに命令を与えて並列プログラムを制御させ実行制御結果を取得し、さらに取得した各逐次プログラムの実行制御結果に基づいて命令を与えるようにしたので、複数の逐次プログラム相互間の動作状態を関連付けて、逐次プログラムの実行制御および変数値の取得など所望のデバッグを行うことができる。

【0140】例えば、複数の逐次プログラム間である条件が成立した場合に、自動的にプログラムを停止させるような処理などが可能となる。

(2) 本発明では、デバッガコントローラ内には逐次デバッガ夫々から取得すべき変数および該変数を取得する逐次プログラム中の位置などを予め設定できるようにし、表示手段内には表示すべき変数や該変数を用いた関数を定義した表示式を予め設定できるようにし、表示手段はこの表示式に従ってデバッガコントローラから与えられた変数の値を表示するようにしたので、並列プログラム内に可視化のための関数などをプログラムに埋め込むなどして並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる。

【図面の簡単な説明】

【図1】 本発明の第1の実施例に係る並列プログラムデバッガの構成を示す図

【図2】 デバッグコントローラ内での処理の手順を示すフローチャート

【図3】 2箇所のブレークポイントを設定されたプログラムの一例を示す図

【図4】 実行制御命令記述ファイルの一例を示す図

【図5】 変数が後の命令で破壊されるプログラムの一例を示す図

【図6】 3つの収束ループを持つプログラムの一例を示す図

【図7】 5つのプログラムのブレークポイントでの停止状況を示す図

【図8】 デバッグコントローラ内に保持されるデータ構造の一例を示す図

【図9】 本発明の第2の実施例に係る並列プログラム可視化装置の構成を示す図

【図10】 2次元表示を行う表示部の一例を示す図

【図11】 ブローポイントの登録にあたって変数取得表に記憶される内容を示す図

【図12】 変数記憶表の一例を示す図

【図13】 ショウポイントの登録にあたって変数取得表に記憶される内容を示す図

【図14】 2次元グラフの描画例を示す図

【図15】 プロセッサ番号とブレークポイントのグラフの表示例を示す図

【図16】 テンプレート変数への式を記述した例を示す図

【図17】 本発明の第3の実施例に係る並列プログラム可視化装置の構成を示す図

【図18】 簡易言語の記述例を示す図

【図19】 変数の過去の値を利用して表示を行った例を示す図

【図20】 本発明の第4の実施例に係る並列プログラム可視化装置の構成を示す図

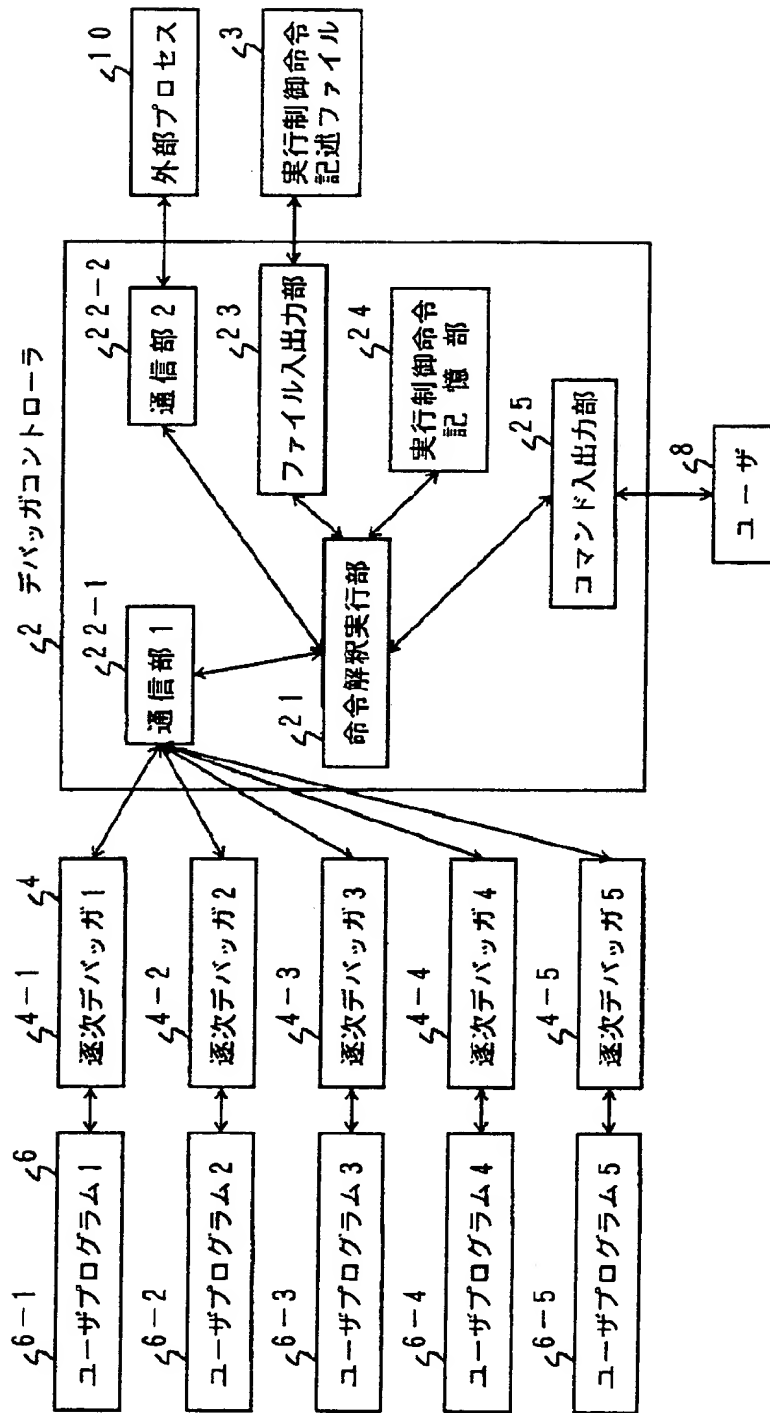
【図21】 表示部での図形データの記憶方式を説明するための図

【図22】 表示部上の図形に関係するプロセッサの番号を表示する例を示す図

【符号の説明】

2…デバッガコントローラ、3…実行制御命令記述ファイル、4、4-1～4-5…逐次デバッガ、6、6-1～6-5…ユーザプログラム、8…ユーザ、10…外部プロセス、21…命令解釈実行部、22-1…第1の通信部、22-2…第2の通信部、23…ファイル入出力部、24…実行制御命令記憶部、25…コマンド入出力部、102…デバッガコントローラ、104、104-1～104-n…逐次デバッガ、106、106-1～106-n…ユーザプログラム、107、107-1～107-n…表示部、108…ユーザ、121…制御部、122-1…第1の通信部、122-2…第2の通信部、125…ユーザインタフェース、131…変数取得表、132…変数記憶表、140…通信部、141…可視化処理部、142…テンプレート変数記述部、143…式解釈部、144…描画部、160…簡易言語処理部、161…簡易言語記憶部、162…簡易言語解釈実行部、163…簡易言語変数記憶部、207…表示部、210…ユーザイベントハンドラ、211…図形データ記憶部、212…プロセッサマップ表示部

【図1】



【図3】

```

00      while (1) {
01          if (y>100)
02      1      y=f(x, y);
03          else
04      2      y=g(x, y);
05      }

```

【図4】

```

while (1) {
    $BP1NUM=0;
    for ($i=0; $i<$PENUM;$i++) {
        if (BP($i)==1) $BP1NUM++;
    }
    if ($BP1NUM==2)
        break;
    WAIT (#PEALL);
}

```

【図5】

```

00      for (i=0; i<100; i++) {
01          x=f(y);
02      P      ...;
03          x=x+b;
04      B      y=g(x);
05      }

```

【図7】

回数	停止したブレークポイント番号				
	PE1	PE2	PE3	PE4	PE5
1	2	1	1	1	1
2	1	2	1	2	1
3	1	3	3	1	2
4	3			3	1
5				3	
6	1	4	2	1	1
7	2		1	2	2
8	1		3	3	1

【図6】

```

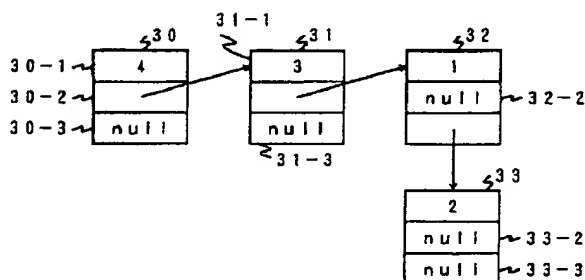
00      for (i=0; i<100; i++) {          /*収束ループ1*/
01          while (e<0.1) {              /*収束ループ2*/
02              while (dx<0.1) {          /*収束ループ3*/
03                  if (y>0) {
04      1                      x=...;
05                  } else {
06      2                      x=...;
07                  }
08              }
09      3          e=sqrt((y1-y)^2*(x1-x)^2);
10              }
11      4          ee=e;
12      }

```

【図12】

行番号	プログラム番号	変数リスト
10	0	x=1
10	0	y=5
10	1	x=10
10	1	y=18
10	2	x=3
10	2	y=8
10	3	x=7
10	3	y=12
10	4	x=20
10	4	y=4

【図8】



【図11】

行番号	フラグ	変数リスト
10	P	x, y

【図18】

```

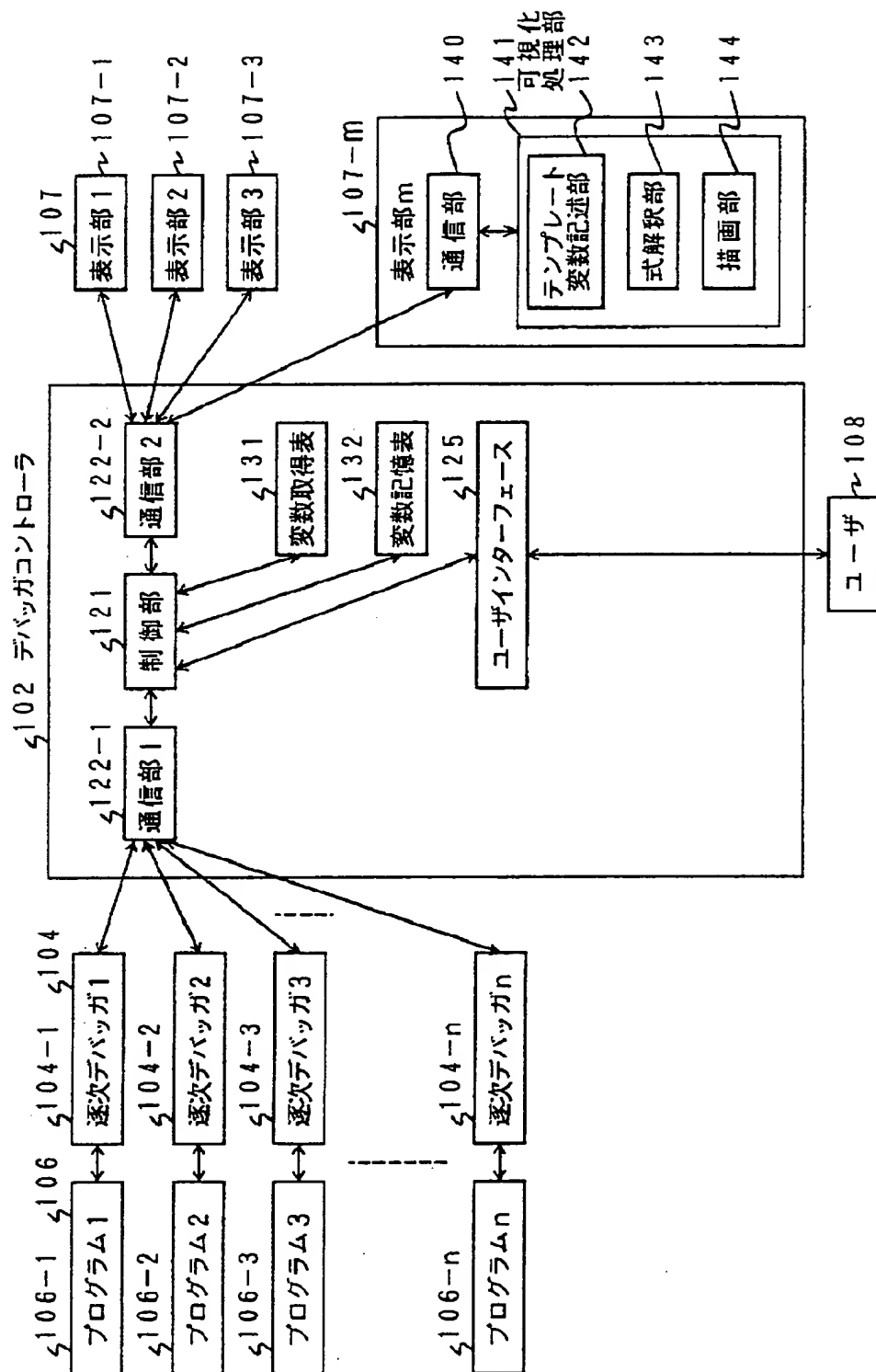
01  extern x, y
02  view 2
03  oldx = x;
04  oldy = y;

```

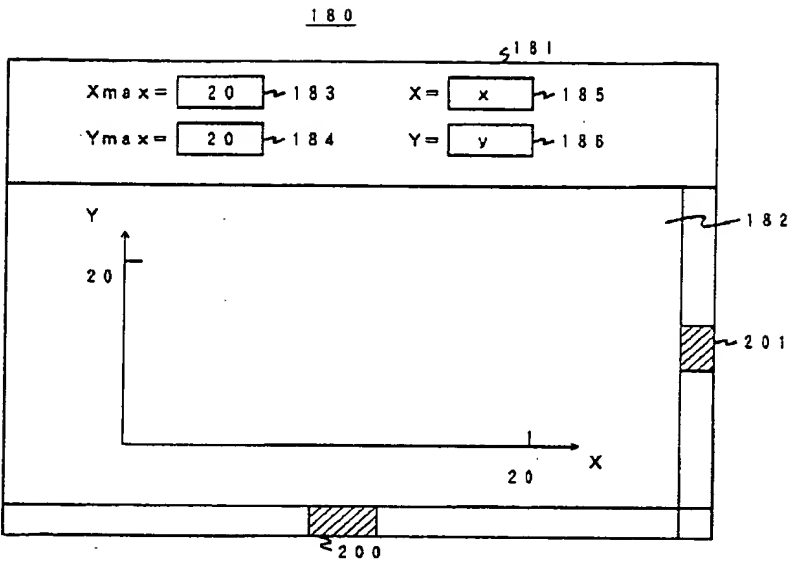
【図13】

行番号	フラグ	変数リスト
11	S	

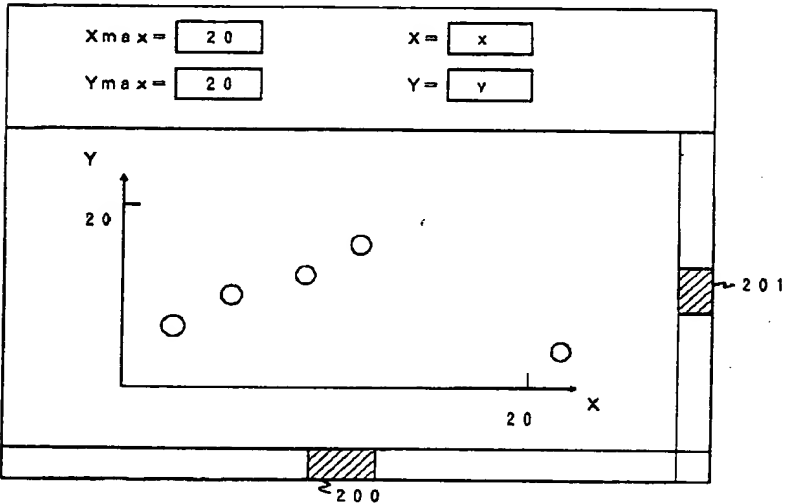
【図9】



【図10】



【図14】



【図20】

【図21】

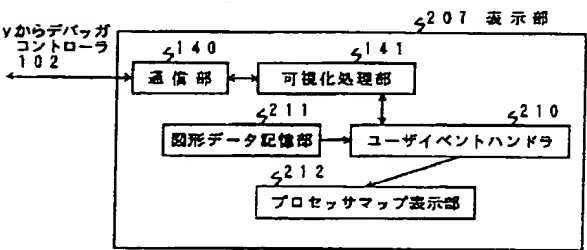
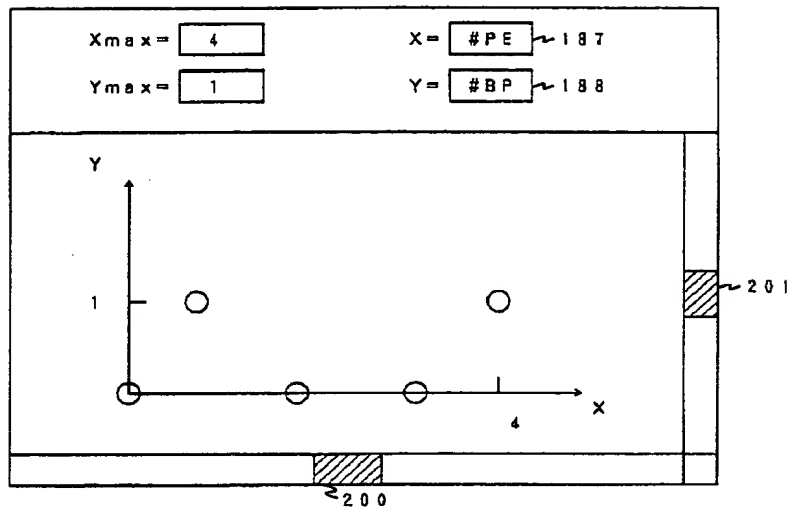
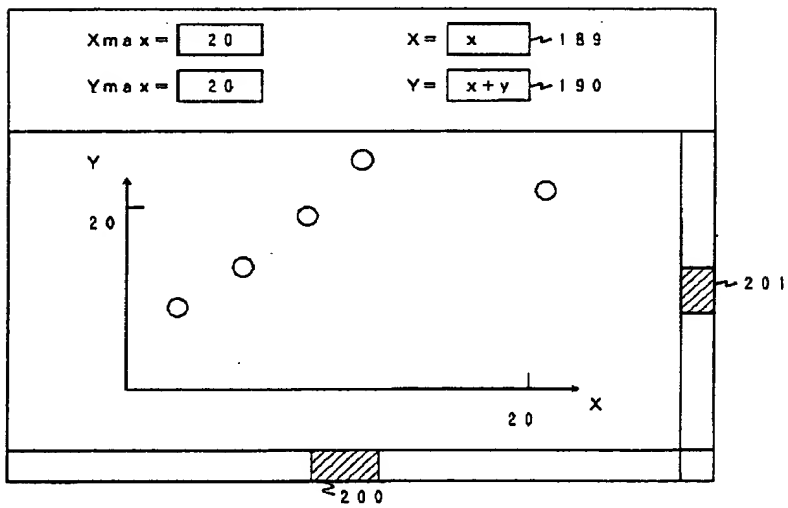


図 号	mouse X	mouse Y	変 数 リスト	プロセッサ 番 号	ショウ ポイント番号
1	100	500	x=1 y=5	0	1
2	1000	1800	x=10 y=18	1	1
3	300	800	x=3 y=8	2	1
4	700	1200	x=7 y=12	3	1
5	2000	400	x=20 y=4	4	1

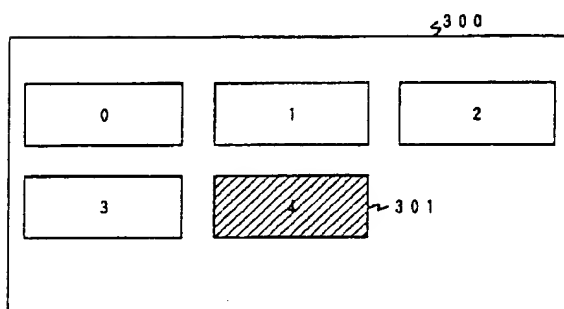
【図15】



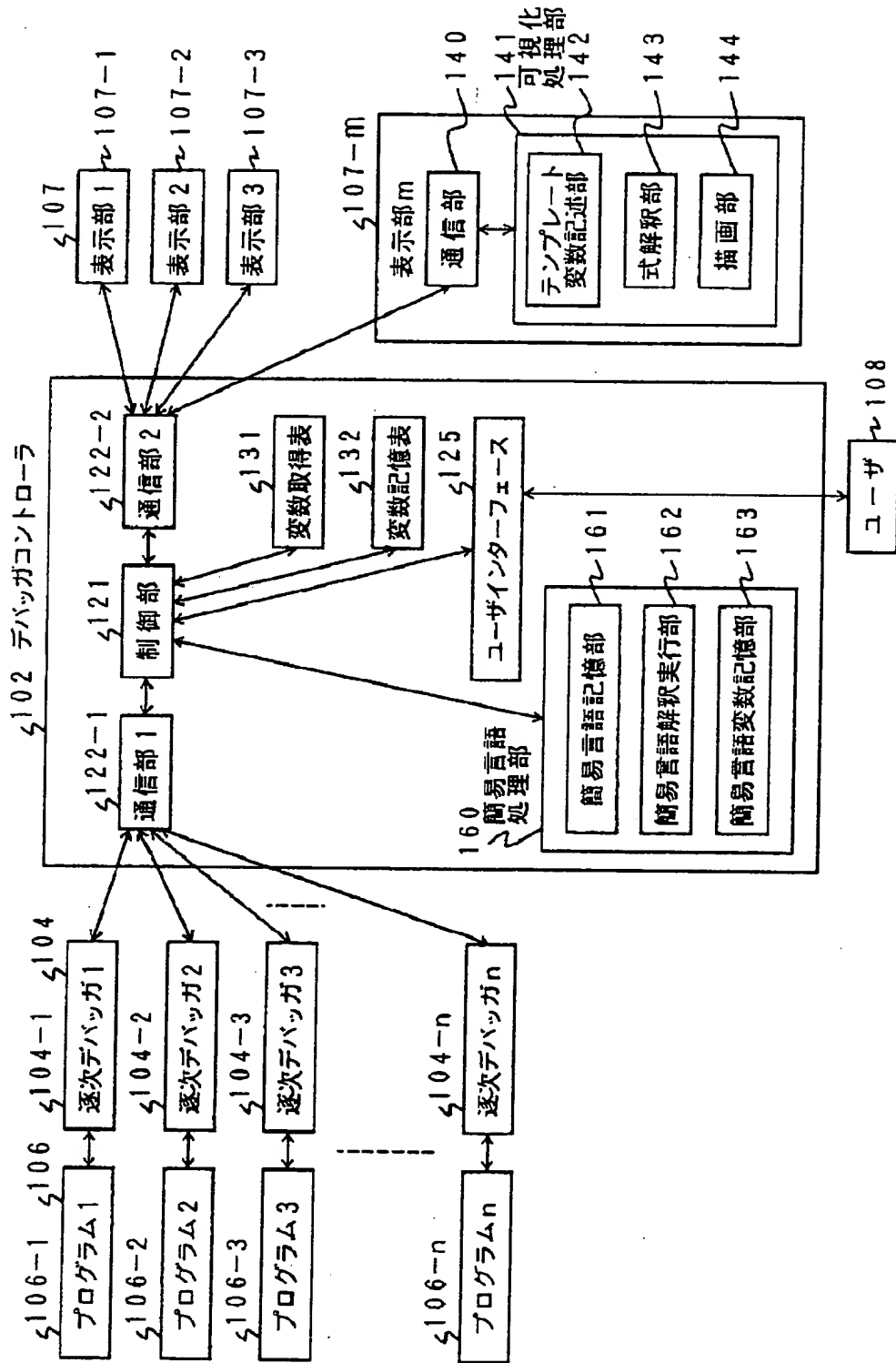
【図16】



【図22】



【図17】



【図19】

